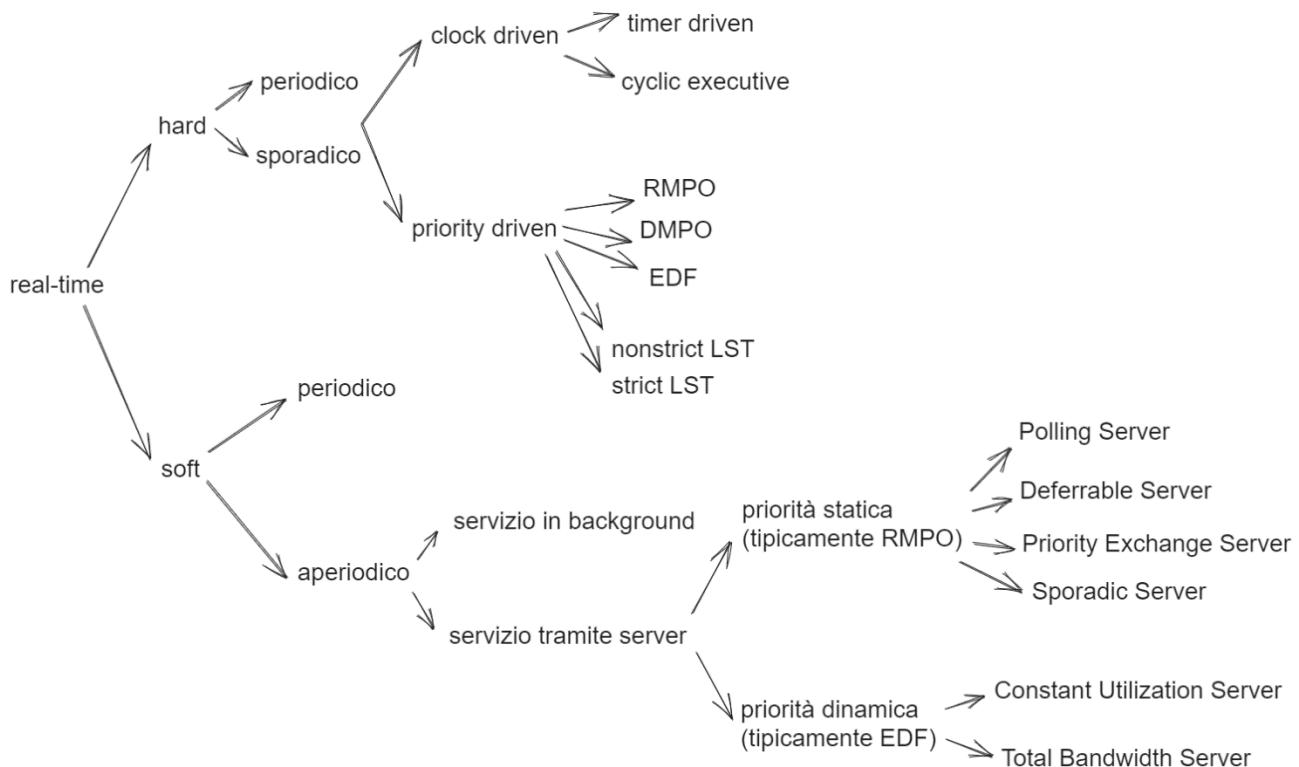


SRT Guida V3



PROBLEMA 1

- **RELAZIONE ARMONICA:** se presa qualunque coppia di periodi, il secondo della coppia è multiplo del primo.

E' IL PIU' PICCOLO NUMERO DIVISIBILE PER ENTRAMBI I NUMERI DATI.

→ **REGOLA:** SI TROVA MOLTIPLICANDO I FATTORI PRIMI, COMUNI E NON COMUNI, PRESI UNA SOLA VOLTA, CON L'ESPONENTE PIU' GRANDE.

Procedimento:

- 1) SCOMPORRE IN FATTORI PRIMI I NUMERI
- 2) MOLTIPLICARE SEGUENDO LA REGOLA I FATTORI

Esempio :

m.c.m. (14 , 24 , 36)

14	2	24	2	36	2
7	7	12	2	18	2
1		6	2	9	3
		3	3	3	3
		1		1	

$$\begin{aligned} 14 &= 2 \times 7 \\ 24 &= 2^3 \times 3 \\ 36 &= 2^2 \times 3^2 \end{aligned}$$

SCELGO I FATTORI, COMUNI E NON COMUNI CON L'ESPONENTE PIU' ALTO



$$\text{m.c.m.} = 7 \times 2^3 \times 3^2 = 7 \times 8 \times 9 = 504$$

E' IL PIU' GRANDE TRA I DIVISORI COMUNI DI DUE O PIU' NUMERI.

→ **REGOLA:** SI TROVA MOLTIPLICANDO I FATTORI PRIMI, COMUNI PRESI UNA SOLA VOLTA CON L'ESPONENTE PIU' PICCOLO.

Procedimento:

- 1) SCOMPORRE IN FATTORI PRIMI I NUMERI
- 2) MOLTIPLICARE SEGUENDO LA REGOLA I FATTORI

Esempio :

M.C.D. (12 , 24 , 36)

12	2	24	2	36	2
6	2	12	2	18	2
3	3	6	2	9	3
1		3	3	3	3
		1		1	

$$\begin{aligned} 12 &= 2^2 \times 3 \\ 24 &= 2^3 \times 3 \\ 36 &= 2^2 \times 3^2 \end{aligned}$$



SCELGO I FATTORI CHE I NUMERI HANNO IN COMUNE, CON L'ESPONENTE PIU' PICCOLO.

$$\text{M.C.D.} = 2^2 \times 3 = 4 \times 3 = 12$$

Prerequisiti

Vengono forniti dal testo un certo numero di processi Pi:

1. Aggiungere la colonna U_i calcolata come C_i/T_i
2. Calcolare U_p come la **somma di tutti gli U_i** calcolati sopra.
3. Se D_i non viene fornito dal testo, allora $D_i = T_i$
4. Aggiungere la colonna $\delta_i = D_i/T_i$

	T_i [t.u.]	C_i [t.u.]
P ₁	10	2
P ₂	15	4
P ₃	20	5
P ₄	30	3
P ₅	60	2

A) Schedulazione Clock Driven – approccio Cyclic Executive

a) dimensionamento del ciclo maggiore “M” (iperperiodo)

M = mcm (T₁, ..., T_N) [minimo comune multiplo]

b) dimensionamento del ciclo minore “m”

Restringere l'insieme dei possibili “m” applicando le seguenti proprietà.

- $m \geq C_i \quad \forall i$
- $m \leq T_i \quad \forall i$
- **M mod m = 0** [resto della divisione uguale a zero]
- **2*m - MCD(m, T_i) <= T_i $\forall i | (T_i \text{ mod } m) > 0$**
Parti a calcolare dall’“m” più elevato possibile e fermati al primo ammissibile, poiché nel caso risultassero ammissibili più alternative di “m”, selezionare quello con la dimensione massima.

c) calcolo del numero dei cicli minori e dei job per processo nell’ambito di ciascun ciclo maggiore

- numero cicli minori:

$$n_{cm} = M / m$$

- numero dei cicli dei job in un ciclo maggiore:

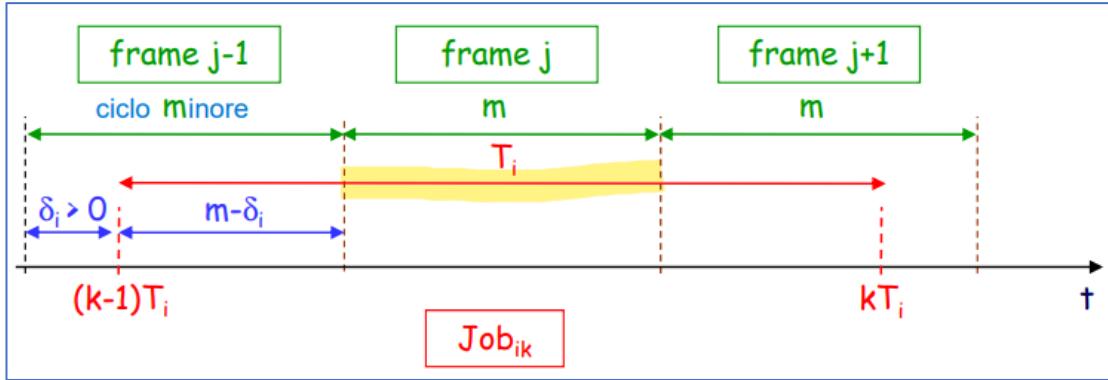
$$n_{j1} = M / T_1$$

$$n_{j2} = M / T_2$$

...

d) identificazione del ciclo minore (o dei cicli minori) in cui ciascun job può essere eseguito

- le righe rappresentano tutti i cicli minori (di dimensione m t.u.) dentro un ciclo maggiore M. Il numero di righe è pari a n_{cm}
- le colonne rappresentano tutti i job dei processi in un ciclo maggiore. Ogni processo J_k ha n_{jk} job dentro un ciclo maggiore. Rappresentarli in ordine da J₁ a J_N
- Disegnare a sinistra della tabella la linea temporale per mettere in chiaro i cicli minori rispetto al trascorrere del tempo.
- Procedi al completamento della tabella considerando un job alla volta. Crocetta sempre il primo ciclo minore per ogni job (J_{x1}). Ogni k-esima iterazione del job deve eseguire entro k*T. Dentro a questo intervallo, che inizialmente è **[0, T]** poi **[T, 2*T]** ecc..., devi crocettare quali cicli minori sono candidati ad ospitare l'esecuzione della k-esima iterazione del job attualmente analizzato (di durata C_i) fintantoché il ciclo minore sia completamente compreso nell'intervallo di tempo evidenziato prima.



	C	T
P_1	1	4
P_2	2	5
P_3	1	10
P_4	2	20

$M = 20, m = 2 \Rightarrow n_{cm} = 10 \quad n_{J1} = 5 \quad n_{J2} = 4 \quad n_{J3} = 2 \quad n_{J4} = 1$

identificazione del ciclo minore (o dei cicli minori) in cui ciascun job può essere eseguito:

	J_{11}	J_{12}	J_{13}	J_{14}	J_{15}	J_{21}	J_{22}	J_{23}	J_{24}	J_{31}	J_{32}	J_{41}
c_1	x					x				x		x
c_2	x						x			x		x
c_3		x								x		x
c_4		x					x			x		x
c_5			x			x				x		x
c_6			x				x				x	x
c_7				x				x			x	x
c_8				x							x	x
c_9					x				x		x	x
c_{10}					x				x		x	x

$0 \downarrow t \quad 2 \quad 4 \quad 6 \quad 8 \quad 10 \quad 12 \quad 14 \quad 16 \quad 18 \quad 20$

e) pianificazione dell'esecuzione di ciascun job

- le righe rappresentano tutti i n_{cm} cicli minori
- le colonne rappresentano la dimensione m dei cicli minori
- Lo scorrere del tempo avviene quindi da sinistra a destra, dall'alto al basso
- Devi inserire tutte le istanze di tutti i job nei cicli minori candidati per quel job (individuati nella tabella precedente) dando la precedenza ai:
 - ai cicli minori con minor tempo residuo libero (cioè fare in modo che il ciclo minore sia completamente pieno, anche a discapito dei due criteri sotto)
 - job di processi con frequenza di esecuzione più elevata
 - job con tempo di esecuzione più elevato
- Segnare a matita sulla tabella precedente i job man mano inseriti, in modo da non dimenticarne!
- Se sono stati inseriti tutti i job, l'esito è positivo.
Altrimenti è negativo, indicando quali job non sono riusciti ad essere inseriti.

	T_i [t.u.]	C_i [t.u.]
P_1	10	2
P_2	15	4
P_3	20	5
P_4	30	3
P_5	60	2

	J_{11}	J_{12}	J_{13}	J_{14}	J_{15}	J_{16}	J_{21}	J_{22}	J_{23}	J_{24}	J_{31}	J_{32}	J_{33}	J_{41}	J_{42}	J_{51}
c_1	x						x				x		x	x		x
c_2		x									x		x	x		x
c_3			x					x				x		x		x
c_4				x					x			x		x	x	x
c_5					x							x		x	x	x
c_6						x				x			x		x	x

- (e) pianificazione dell'esecuzione di ciascun job

c_1	J_{11}	J_{21}	J_{51}		
c_2	J_{12}	J_{31}		J_{41}	
c_3	J_{13}	J_{22}			
c_4	J_{14}	J_{23}			
c_5	J_{15}	J_{33}		J_{42}	
c_6	J_{16}	J_{24}			

esito positivo per tutti i job

sì	no	quali job ? J_{32}
----	----	-------------------------

f) In caso di esito negativo, effettuare il rilassamento del minor numero possibile di vincoli

- Effettuare Job-slicing al processo che non è stato schedulato. Il T rimane uguale, ma viene spezzata la C in due valori tali che i due nuovi job riescano ad essere entrambi schedulati. Aggiungi anche una priorità tra i due.

(f) in caso di esito negativo, rilassamento del minor numero possibile di vincoli

$$P_3 (5, 20) \Rightarrow P_3' (3, 20), P_3'' (2, 20) \text{ con } P_3' \prec P_3''$$

e iterazione del procedimento

c_1	J_{11}	J_{21}	J_{51}		
c_2	J_{12}	J_{31}		J_{41}	
c_3	J_{13}	J_{22}	J_{32}'		
c_4	J_{14}	J_{23}	J_{32}''		
c_5	J_{15}	J_{33}		J_{42}	
c_6	J_{16}	J_{24}			

NB2: si può anche dividere in modo tale che 3 job da 4 diventino 4 job da 3

NB3: più job dello stesso processo che devono essere divisi, vanno suddivisi nella stessa maniera (ad esempio: se mi rimangono fuori J_{71} e J_{72} con $C_i = 10$, non posso suddividerli a $J_{71}' = 5$, $J_{71}'' = 5$ e $J_{72}' = 7$ e $J_{72}'' = 3$, ma o sono entrambi 5/5 o sono entrambi 7/3. Ricorda che è lo stesso processo quello che stai suddividendo)

B) Schedulazione Priority Driven

Test basato sul fattore di utilizzazione del processore (qualsiasi) → necessaria (se fallisce è conclusivo, se positivo è non conclusivo)

Affinché un insieme di N processi periodici e sporadici sia schedulabile è che:

$$U = \sum_{j=1}^N U_j = \sum_{j=1}^N \frac{C_j}{T_j} \leq 1$$

Teorema di Liu-Layland (RMPO) → sufficiente

Un insieme di N processi è schedulabile con RMPO se:

$$U \leq U_{RMPO}(N) = N(2^{1/N} - 1)$$

Corollario del Teorema di Liu-Layland (RMPO) → sufficiente

Un insieme di N processi è schedulabile con RMPO se:

$$U_{RMPO}(U_1, U_2, \dots, U_N) = \prod_{j=1}^N (1 + U_j) \leq 2$$

Test di Burchard (RMPO) → sufficiente

Un insieme di N processi è schedulabile con RMPO se il suo fattore di utilizzazione $U \leq U_{RMPO}$ dove U_{RMPO} è calcolato come segue:

$$X_j = \log_2 T_j - \lfloor \log_2 T_j \rfloor, \quad \forall j$$

$$\zeta = \max_{1 \leq j \leq N} X_j - \min_{1 \leq j \leq N} X_j$$

$$U_{RMPO}(N, \zeta) = \begin{cases} (N-1)(2^{\frac{\zeta}{(N-1)}} - 1) + 2^{1-\zeta} - 1 & \zeta < 1 - \frac{1}{N} \\ N(2^{\frac{1}{N}} - 1) & \zeta \geq 1 - \frac{1}{N} \end{cases}$$

Test di Han (RMPO) → sufficiente

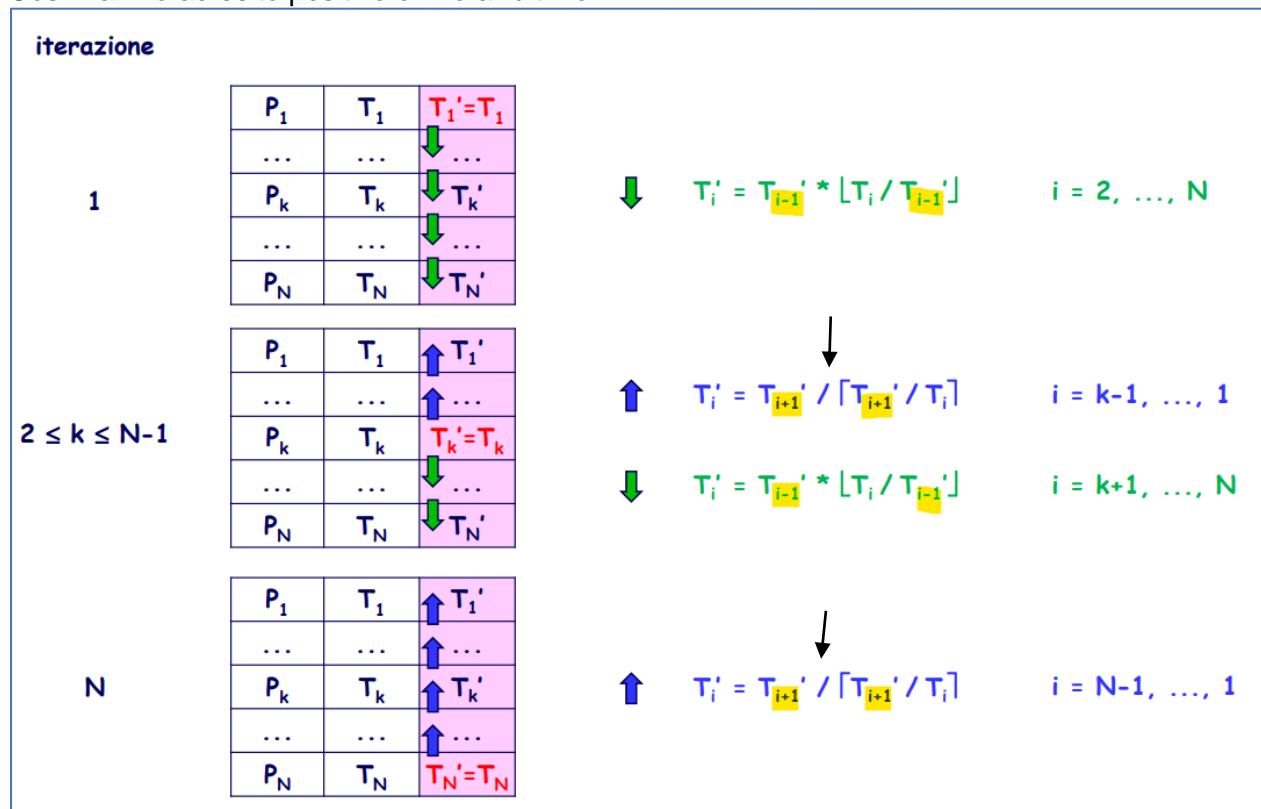
Un insieme di N processi P₁, P₂, ..., P_N è schedulabile con RMPO se ad esso corrisponde un insieme “accelerato” (ossia diminuiscono i T) di N processi P_{1'}, P_{2'}, ..., P_{N'} semplicemente periodici e con fattore di utilizzazione del processore $U' = U_1' + U_2' + \dots + U_N' \leq 1$.

I processi si accellerano in modo tale che siano in relazione armonica tra loro, lasciando invariato C.

Si parte dal primo periodo: si tiene fermo T₁ e si modificano gli altri in discesa affinché siano tutti in relazione armonica.

Se l'esito non è positivo ($U' > 1$), si passa alla seconda iterazione in cui si ripristina il valore di (solo) T₂, poi si modifica prima verso l'alto poi verso il basso.

Così via fino ad esito positivo o fino all'ultimo T_n.



Se **SALI** approssimi per **ECCESSO** (^ parte **BLU**)

Se **SCENDI** approssimi per **DIFETTO** (v parte **VERDE**)

Test di Kuo-Mok (RMPO) → sufficiente

Un insieme S di N processi P_1, P_2, \dots, P_N contraddistinto da un fattore di utilizzazione del processore U è schedulabile con RMPO se $U \leq U_{\text{RMPO}}(K)$, essendo K il numero di sottoinsiemi disgiunti di processi semplicemente periodici in S .

La schedulabilità dell'insieme S è garantita se è schedulabile un insieme S' di $K < N$ processi P'_1, P'_2, \dots, P'_K ad esso equivalente. Al processo P'_j ($j = 1, 2, \dots, K$) in S' , corrispondente al sottoinsieme di processi semplicemente periodici P_x, P_y, \dots, P_z in S , compete un fattore di utilizzazione del processore $U'_j = U_x + U_y + \dots + U_z$, un periodo $T'_j = \min(T_x, T_y, \dots, T_z)$ e un tempo di esecuzione $C'_j = U'_j T'_j$.

- 1) Raggruppare i processi in relazione armonica tra loro

S	C	T	C/T
P_1	4	10	0.40
P_2	4	20	0.20
P_3	8	40	0.20
P_4	3.6	45	0.08
P_5	1.8	90	0.02

kuo mok

$$P'_1 \equiv \{P_1, P_2, P_3\}$$

$$P'_2 \equiv \{P_4, P_5\}$$

S'	C'	T'	C'/T'
P'_1	8	10	0.8
P'_2	4.5	45	0.1

- 2) Se un processo può stare in più raggruppamenti, lo si inserisce nel gruppo che ha fattore di utilizzazione più alto (prima del suo inserimento), in modo tale che i fattori U finali siano più diversi possibile e che quindi il corollario L-L possa dare risultato positivo.
- 3) Ogni nuovo processo ha
 - Periodo uguale al minimo periodo tra quelli che ha raggruppato
 - Fattore di utilizzazione uguale alla somma dei fattori di utilizzazione dei processi che ha raggruppato
 - Computation time uguale al prodotto tra il suo fattore di utilizzazione per il suo periodo.
- 4) Se non è richiesto di applicare il corollario di Liu-Layland, applicare il teorema di Liu-Layland.

Test basato sul calcolo della massima interferenza che ogni processo può subire entro la deadline specificata (RMPO) → sufficiente

$$C_i + \sum_{j \mid p_j > p_i} \left\lceil \frac{T_i}{T_j} \right\rceil C_j \leq T_i \quad i = 1, 2, \dots, N$$

Algoritmo di Audsley (**RMPO**) → necessario e sufficiente

- 1) La tabella è del tipo:

	P ₁	P ₂	P ₃	P ₄
R _i ⁰				
R _i ¹				
R _i ²				
R _i ³				
R _i ⁴				
R _i ⁵				

In cui ogni riga è una iterazione dei processi. Ad esempio la cella in alto a sinistra è sempre il tempo di risposta R_i⁰

- 2) Procedi a calcolare colonna per colonna a partire da P₁. Ti fermi a calcolare una colonna appena trovi due tempi di risposta uguali, quindi passi alla colonna successiva.

$$R_i^0 = C_i \quad \forall i$$

$$R_i^1 = R_i^0$$

$$R_i^n = C_i + \sum_{j | p_j > p_i} \left\lceil \frac{R_i^{n-1}}{T_j} \right\rceil C_j$$

- 3) È schedulabile con RMPO se il tempo finale di risposta di ogni processo è $\leq T_i$.

Algoritmo di Audsley (**DMPO**) → necessario e sufficiente

L'unica differenza con RMPO è che i tempi di risposta devono essere $\leq D_i$

Test basato sulla densità di utilizzazione del processore (**DMPO**) → sufficiente

Affinché un insieme di N processi periodici e sporadici sia schedulabile con l'algoritmo DMPO è che:

$$\Delta = \sum_{j=1}^N \frac{C_j}{D_j} \leq U_{RMPO}(N) = N(2^{1/N} - 1)$$

Test di Lehoczky (DMPO) → sufficiente

Un insieme di N processi periodici e sporadici, ciascuno contraddistinto da una deadline relativa $D_j = \delta_j * T_j$, $j = 1, 2, \dots, N$, sia schedulabile con l'algoritmo DMPO è che:

$$\sum_{j=1}^N \frac{C_j}{T_j} \leq U(N, \delta) = \begin{cases} N((2\delta)^{1/N} - 1) + 1 - \delta & 0.5 \leq \delta \leq 1 \\ \delta & 0 \leq \delta \leq 0.5 \end{cases} \quad \delta = \min_j \{\delta_j\}$$

Test basato sul fattore di utilizzazione efficace del processore (DMPO) → sufficiente

- 1) Usare le deadline fornite o, in loro assenza, i rispettivi periodi.
- 2) Da Lehoczky: $\delta_i = D_i / T_i$
- 3) Scrivere nella tabella i processi in ordine di priorità (il più alto ha deadline più bassa)
- 4) Calcolare il “fattore di utilizzazione efficace”

$$f_j = \left(\sum_{i \in H_n} \frac{C_i}{T_i} \right) + \frac{1}{T_j} \left(C_j + \sum_{k \in H_l} C_k \right)$$

- H_n è il sottoinsieme di processi con priorità maggiore al processo considerato P_j (ossia quelli sopra) con $T_i < D_j$ (ATTENZIONE agli indici)
- H_l è il sottoinsieme di processi con priorità maggiore al processo considerato P_j (ossia quelli sopra) con $T_i \geq D_j$ (ATTENZIONE agli indici)
- Per il primo processo sono sempre vuoti {}

- 5) Calcolare:

$$U(N, \delta) = \begin{cases} N((2\delta)^{1/N} - 1) + 1 - \delta & 0.5 \leq \delta \leq 1 \\ \delta & 0 \leq \delta \leq 0.5 \end{cases}$$

- $N = |H_n| + 1$ ovvero la cardinalità di H_n (il numero di elementi) + 1.
- $\delta = \delta_i$

- 6) Il test risulta passato se

$$f_i \leq U(N, \delta) \quad \forall i$$

Diagramma temporale (DMPO) → necessario e sufficiente

Ordinare i processi secondo la priorità DMPO (più la deadline è di valore basso più il processo è prioritario). La priorità rimane statica (fissa) per sempre: non confondere con EDF!

Diagramma temporale (EDF) → necessario e sufficiente

Ad ogni job release (/arrival) viene calcolata la priorità dei processi in base alla deadline assoluta minore, ossia il processo con la deadline più vicina ha la priorità sugli altri. In caso di parità di deadline assolute, si va in base al più grande computation time nominale.

Processor Demand (EDF) → necessario e sufficiente

- 1) Calcolo **Busy Interval**. Proseguì finché $BI^n == BI^{n-1}$. Il Busy Interval finale è l'ultimo calcolato.

$$BI^0 = \sum_{i=1}^N C_i$$

$$BI^n = \sum_{i=1}^N \left\lceil \frac{BI^{n-1}}{T_i} \right\rceil C_i$$

- 2) Calcoli **Hyperperiod = mcm tra i periodi dei processi**. Verifichi che $BI < \text{Hyperperiod}$.
- 3) Calcoli

$$t^* = \frac{\sum_{i=1}^N \left(1 - \frac{D_i}{T_i}\right) C_i}{1 - U}$$

- 4) Calcoli

$$\mathcal{D} = \{d_{ij} \mid d_{ij} = j^* T_i + D_i, d_{ij} < \min(BI, t^*), 1 \leq i \leq N, j \geq 0\}$$

- 5) Calcoli per ogni elemento "t" nell'insieme precedente (in ordine crescente) e per ogni "i" da 1 a N:

$$C_i(0, t) = \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right) C_i$$

$$C_P(0, t) = \sum_{i=1}^N C_i(0, t)$$

	C1(0,t)	...	CN(0,t)	CP(0,t)
t ₁				
...				
t _{ultimo}				

- 6) È schedulabile con EDF se $C_P(0,ti) \leq t_i$ per ogni t (per ogni riga).

Test basato sulla densità di utilizzazione del processore (**EDF**) → sufficiente

Affinché un insieme di N processi periodici e sporadici sia schedulabile è che:

$$\Delta = \sum_{j=1}^N \frac{C_j}{D_j} \leq 1$$

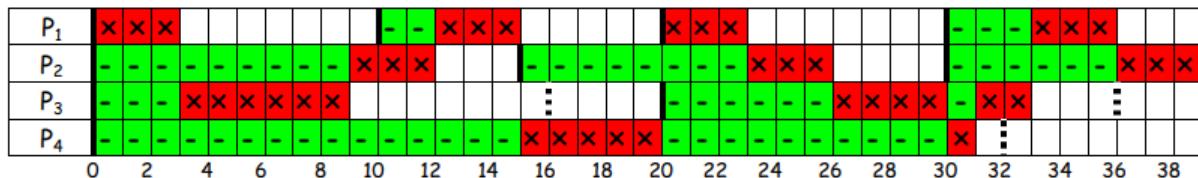
Diagramma temporale (**non-strict LST**) → necessario e sufficiente

Ad ogni job release (/arrival) viene calcolata la priorità dei processi in base allo slack time minore, ossia il processo con slack time più piccolo ha la priorità sugli altri.

Slack-time = prossima deadline assoluta – istante attuale – (computation time – computation time già eseguito nell'istante decisionale)

Se slack-time = 0 allora significa che ha già eseguito tutto e quindi non va contato: mettere un trattino “-“

slack	t=0	t=10	t=15	t=20	t=30
P ₁	7	7	-	7	7
P ₂	12	3	12	7	12
P ₃	10	-	-	10	4
P ₄	26	16	11	11	1



PROBLEMA 2

P_i	R_a	S
Idle	None	Idle
-	-	Ready
...	Pending	Waiting
Running	Being Served	Running

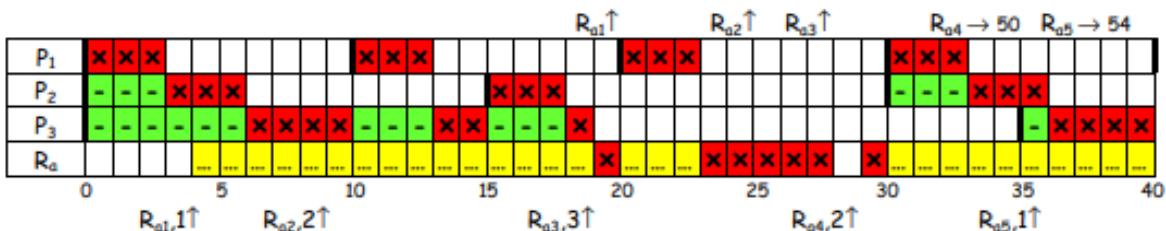
- **RUNNING/SERVED** = processo in esecuzione o richiesta asincrona servita.
- **READY** = Se il processo/server vorrebbe eseguire ma non può eseguire perché stanno eseguendo altri più prioritari.
- **PENDING/WAITING** = server che ATTENDE l'arrivo di una richiesta asincrona. Appena la richiesta arriva si entra nello stato READY.
- **IDLE/NONE** = Quando ho terminato le mie esecuzioni e sono in attesa del prossimo periodo (per un processo) o di una successiva richiesta asincrona (R_a).

Servizio in background

- Le richieste aperiodiche (soft) vengono servite solo se non vi sono processi periodici (o sporadici, entrambi hard) pronti per l'esecuzione.
- I processi aperiodici sono schedulati secondo FCFS (First Come First Served).
- I processi periodici sono schedulati in base al protocollo indicato (tipicamente RMPO, ossia la priorità di ogni processo è statica a priori, ed è inversamente proporzionale al periodo T).
- Segnare i tempi di arrivo e di terminazione dei processi aperiodici, anche se sforano dal diagramma fornito.

	T_i [t.u.]	C_i [t.u.]
P_1	10	3
P_2	15	3
P_3	35	7

	a_i [t.u.]	C_i [t.u.]
R_{a1}	4	1
R_{a2}	9	2
R_{a3}	19	3
R_{a4}	29	2
R_{a5}	36	1



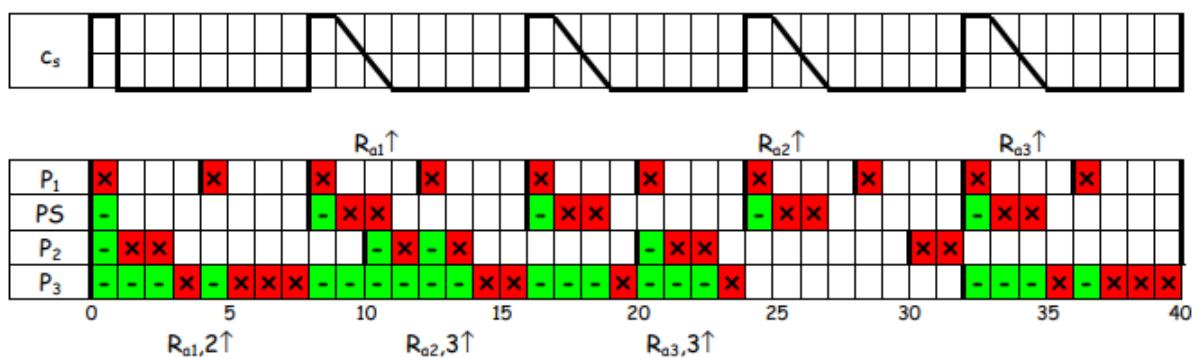
Polling server

- [Server a Priorità Statica]

Il server è un processo periodico con periodo T_s e tempo massimo di esecuzione C_s (capacità).

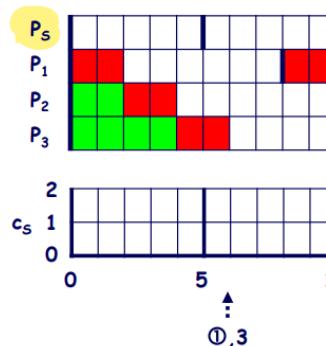
La schedulazione del server segue la stessa strategia utilizzata dai processi periodici (tipicamente RMPO, ossia la priorità di ogni processo è statica a priori, ed è inversamente proporzionale al periodo T).

Quindi bisogna definire la priorità del server rispetto agli altri processi.
- La capacità del server è ripristinata al valore nominale all'inizio di ogni periodo T_s (quindi parte carico).
- La capacità è progressivamente consumata durante il servizio di richieste aperiodiche.
- La capacità viene istantaneamente consumata nei momenti in cui il server è idle (NON quando è ready) ossia quando è il turno del server e non ci sono altre richieste da servire. Quindi non potrà mai essere nello stato **WAITING**.

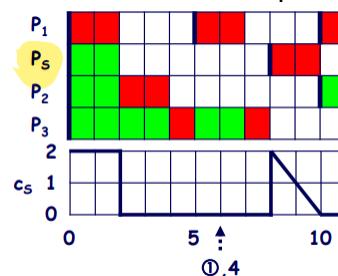


- **ATTENZIONE** a questi casi particolari:

- Il server azzera immediatamente la capacità al tempo 0 poi al tempo 5 perché non sono arrivate risorse.



- Il server azzera la capacità solo quando è in grado di eseguire, non prima



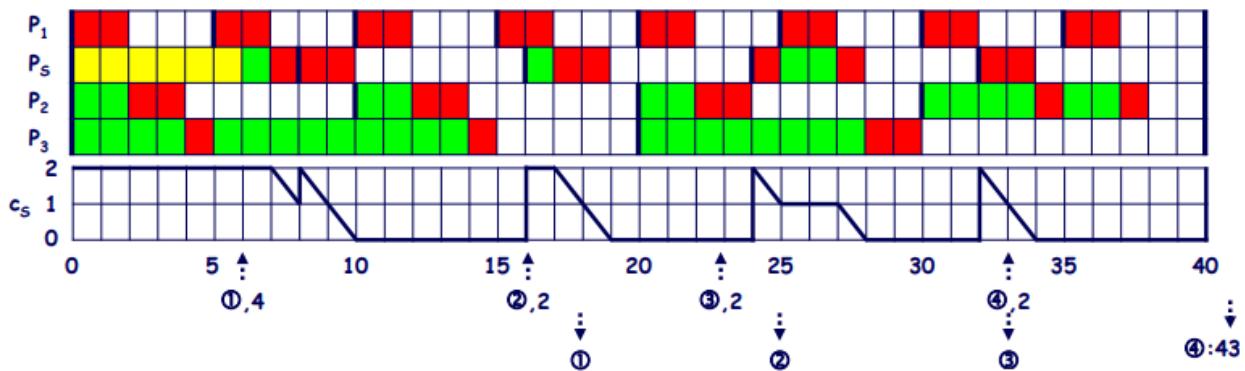
Deferrable Server

- [Server a Priorità Statica]

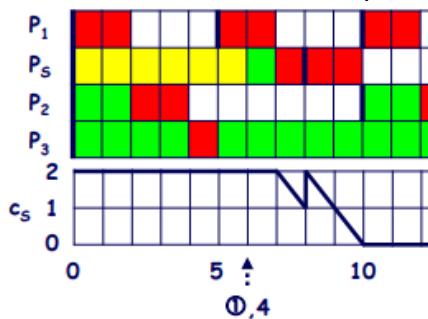
Il server è un processo periodico con periodo T_s e tempo massimo di esecuzione C_s (capacità).

La schedulazione del server segue la stessa strategia utilizzata dai processi periodici (tipicamente RMPO, ossia la priorità di ogni processo è statica a priori, ed è inversamente proporzionale al periodo T).

Quindi bisogna definire la priorità del server rispetto agli altri processi.
- La capacità del server è ripristinata al valore nominale all'inizio di ogni periodo T_s (quindi parte carico).
- La capacità è progressivamente consumata durante il servizio di richieste aperiodiche.
- La capacità disponibile è conservata in assenza di richieste aperiodiche pendenti (stato WAITING).



- ATTENZIONE a questi casi particolari:
 - Il server è WAITING quando non ha richieste in sospeso, e poi READY quando è arrivata la richiesta ma non può eseguire, infine IDLE quando finisce la capacità.



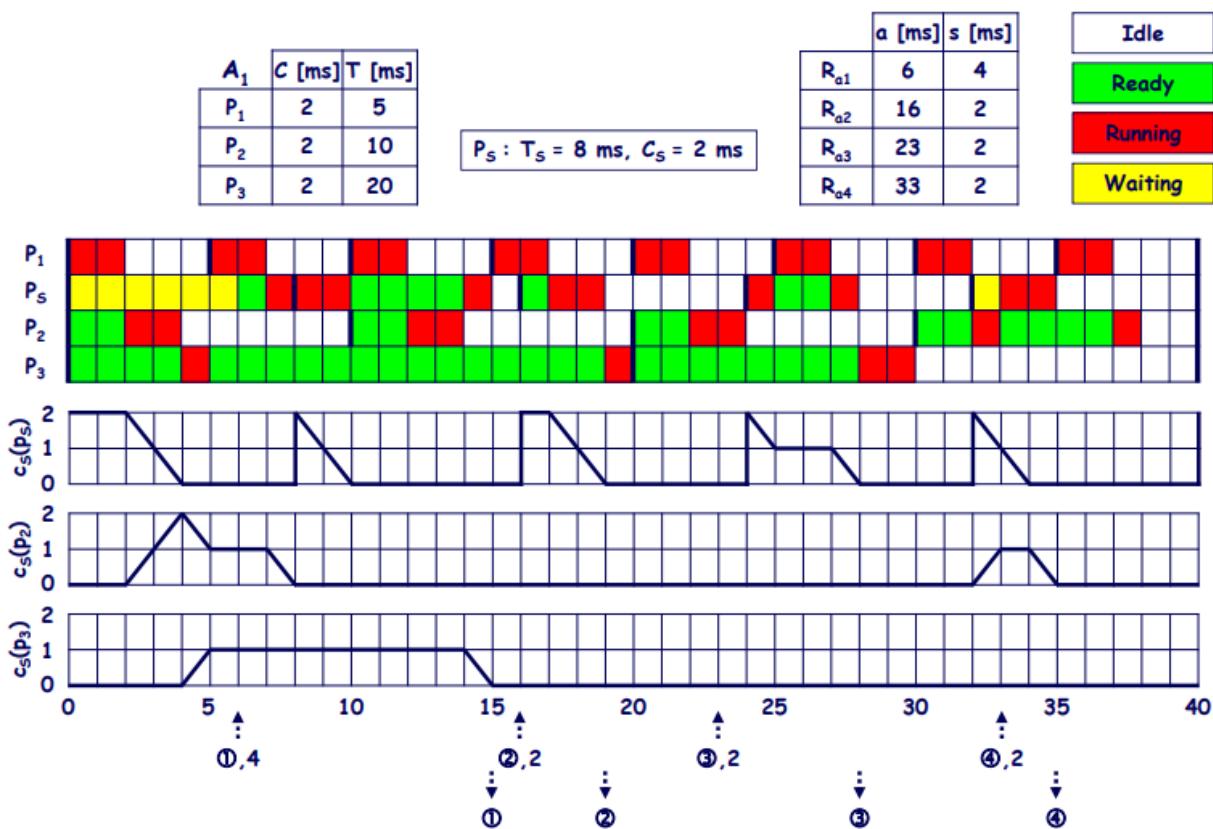
Priority Exchange Server

- [Server a Priorità Statica]

Il server è un processo periodico con periodo T_s e tempo massimo di esecuzione C_s (capacità).

La schedulazione del server segue la stessa strategia utilizzata dai processi periodici (tipicamente RMPO, ossia la priorità di ogni processo è statica a priori, ed è inversamente proporzionale al periodo T).

Quindi bisogna definire la priorità del server rispetto agli altri processi.
- La capacità viene accumulata dal livello di priorità di P_s e dai livelli di priorità dei processi con priorità strettamente inferiore a P_s . Il server è nello stato IDLE solo se in tutti i livelli di priorità non vi è capacità.
- La capacità al livello di priorità $p(P_s)$ è ripristinata al valore nominale all'inizio di ogni periodo T_s (quindi parte carico).
- La capacità disponibile ad un qualunque livello di priorità è conservata durante l'esecuzione di un processo periodico di priorità non inferiore (ossia uguale o maggiore).
- P_s , in presenza di richieste aperiodiche e se dispone di capacità ad un qualche livello di priorità, ha la precedenza rispetto ad un processo periodico di pari priorità.
- La capacità disponibile al massimo livello di priorità è progressivamente consumata:
 - o sia durante il servizio di richieste aperiodiche, se il livello di priorità dove si trova la capacità è maggiore uguale alla priorità degli altri processi pronti
 - o sia in assenza di richieste aperiodiche se non vi sono processi periodici di priorità inferiore in stato ready, ossia se nessuno ha niente da eseguire.
- La capacità disponibile al massimo livello di priorità, in assenza di richieste aperiodiche ed in presenza di almeno un processo periodico P di priorità inferiore che è pronto e sta per eseguire, è progressivamente trasferita al corrispondente livello di priorità del processo P durante l'esecuzione di P , anche se la capacità al livello P è massima.



Sporadic Server

- [Server a Priorità Statica]

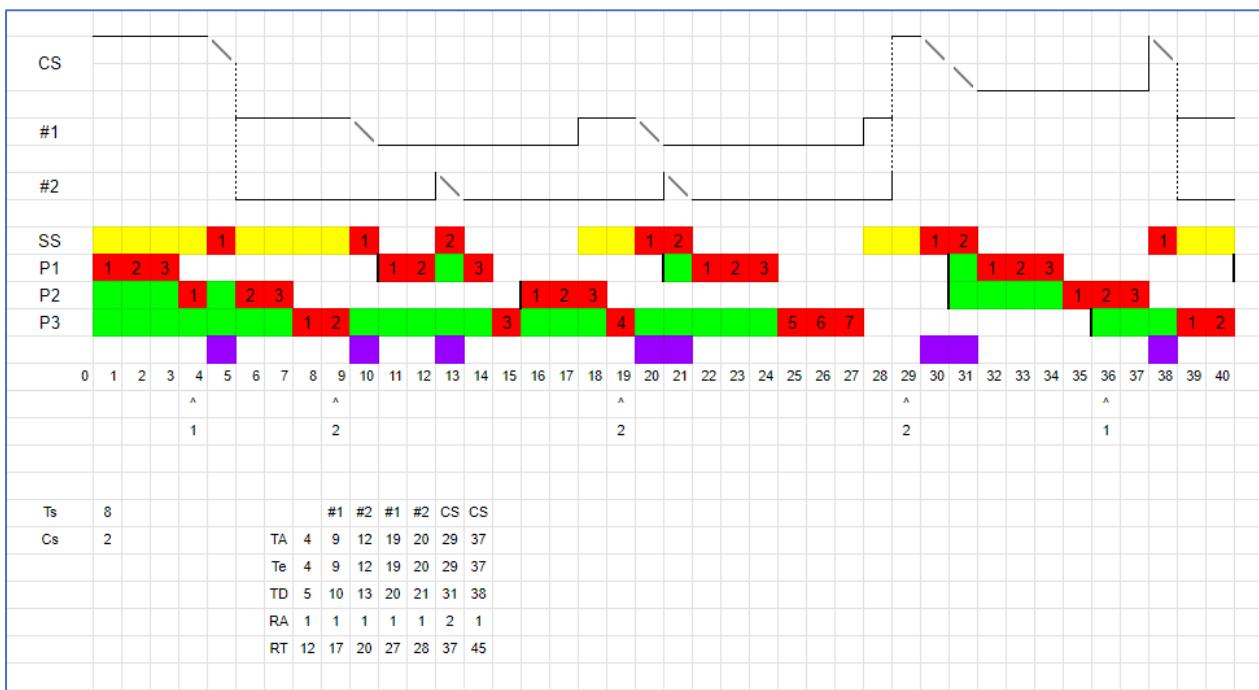
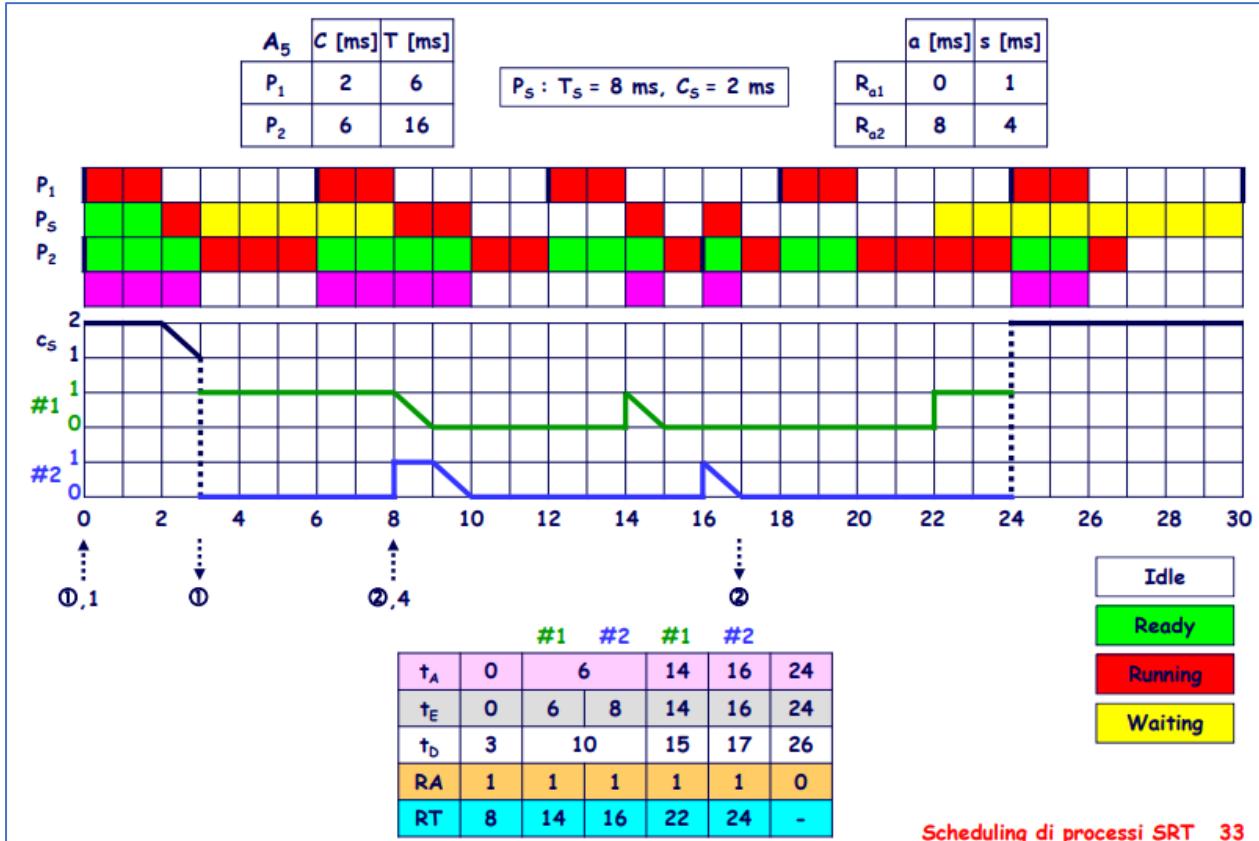
Il server è un processo periodico con periodo T_s e tempo massimo di esecuzione C_s (capacità).

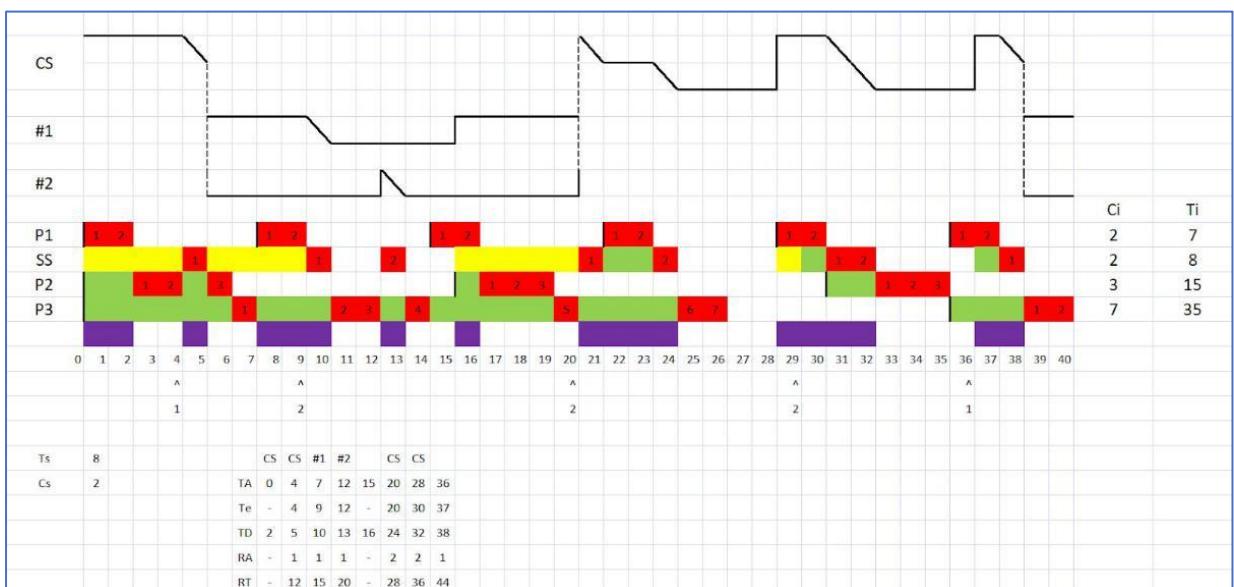
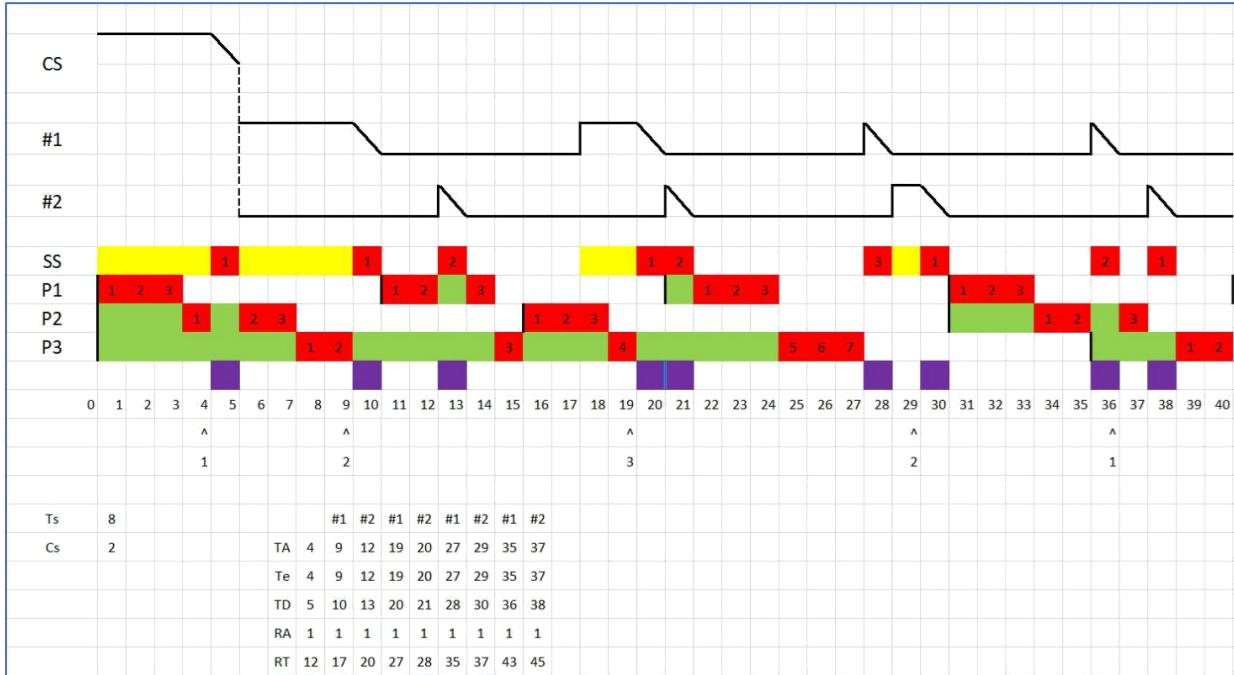
La schedulazione del server segue la stessa strategia utilizzata dai processi periodici (tipicamente RMPO, ossia la priorità di ogni processo è statica a priori, ed è inversamente proporzionale al periodo T).

Quindi bisogna definire la priorità del server rispetto agli altri processi.

- P_s si indica come **attivo** in un dato istante se il processo in esecuzione (RUNNING) ha priorità maggiore uguale al server.

- 1) AD OGNI ISTANTE TI DEVI CHIEDERE SE P_s È ATTIVO E $C_s > 0$ (disegnando dei blocchi viola): in tal caso quello è l'istante t_A .
- 2) APPENA UNA DELLE DUE CONDIZIONI FALLISCE, quello è l'istante t_D , e procedi a calcolare RA e RT.
 - t_A = il successivo primo istante in cui $C_s > 0$ e P_s attivo
 - t_D = il successivo primo istante in cui $C_s = 0$ o P_s non attivo
 - **RA** (replenishment amount) = capacità consumata in $[t_A, t_D]$
 - **RT** (replenishment time) = istante del successivo reintegro = $\max \{t_A + T_s, t_D\}$.
- 3) Se all'istante t_D è rimasta della capacità inutilizzata, si divide in due diverse porzioni (**chunk**) la capacità inutilizzata (#1) e quella consumata (#2).
 - Viene utilizzata la capacità del chunk con RT minore
 - Il RT calcolato precedentemente fa riferimento al #2, mentre si calcola un nuovo RT per #1 quando esso entrerà in un blocco viola.
 - A questo punto si scrivono i valori di #1 e #2 separatamente, sostituendo (?!?) t_A con t_E
 - i. t_E (effective replenishment time) = $\max \{RT, t_A\}$
 - ii. **RA** = capacità consumata in $[t_E, t_D]$
 - iii. **RT** = $\max \{t_E + T_s, t_D\}$
- 4) LA FUSIONE DEI CHUNK AVVIENE APPENA ENTRAMBI HANNO CAPACITA' MASSIMA.
 - La capacità è progressivamente consumata soltanto durante il servizio di richieste aperiodiche. Il server è nello stato IDLE solo se non vi è capacità.
 - La capacità viene reintegrata di quantità RA all'istante RT.
 - La capacità disponibile è conservata in assenza di richieste aperiodiche pendenti (stato WAITING).





Constant Utilization Server

- [Server a Priorità Dinamica]

Il server è un processo sporadico con fattore di utilizzazione $U_s \leq 1 - U_p$.

La schedulazione del server segue la stessa strategia utilizzata dagli altri processi (tipicamente EDF, ossia la priorità di ogni processo è dinamica a runtime, ed è inversamente proporzionale alla deadline assoluta, ossia più la deadline è vicina più è alta la priorità).

Quindi ad ogni job release (/arrival) ed all'arrivo di ogni richiesta aperiodica devi verificare qual è il prossimo processo più prioritario.

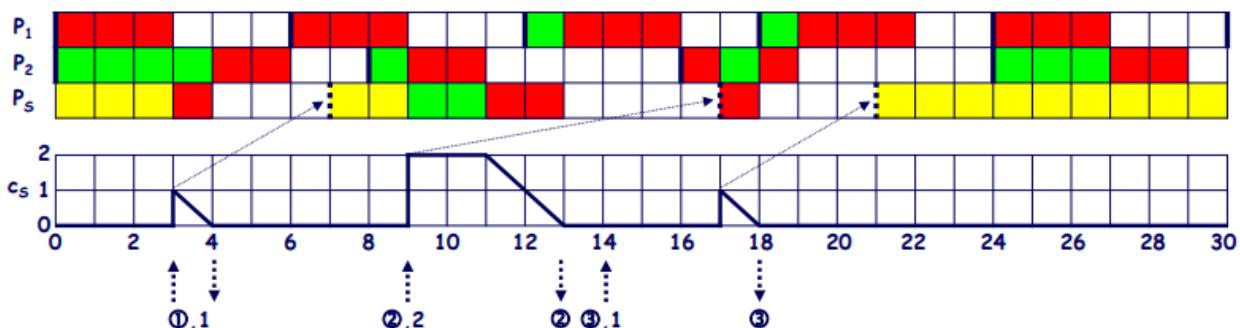
Solitamente non vengono date le deadline dei processi, in quanto si assumono come quelle di default i loro T!

- La capacità del server inizia a zero e lo stato è WAITING.
- La capacità viene consumata progressivamente solo durante il servizio delle richieste aperiodiche.
- Quando arriva una richiesta (al tempo detto t_{Ra}):
 - o la capacità del server c_s si ricarica esattamente di C_{Ra} , ossia il tempo di servizio della richiesta
 - o la deadline relativa diventa $d_s = t_{Ra} + c_s / U_s$
- Le richieste che si presentano prima di d_s vengono accodate.
- Terminata la richiesta da servire (SOLO LA PRIMA), il server è IDLE fino a d_s , dopodiché WAITING fino all'arrivo di una nuova richiesta.
- Se all'istante d_s una o più richieste sono ancora pendenti oppure all'istante di arrivo di una nuova richiesta tale che è superiore a d_s , vengono calcolati la nuova capacità e deadline, e viene posto P_s in stato READY:
 - o Nuovo $c_s = C_{Ra}$
 - o Nuova $d_s = \max(d_s, t_{Ra}) + c_s / U_s$
- In caso il valore calcolato sia decimale si considera $d_s' = \lfloor d_s \rfloor$

A_6	C [ms]	T [ms]	C/T
P_1	3	6	0.50
P_2	2	8	0.25

$$P_s : U_s = 1 - U_p = 0.25$$

a [ms]	s [ms]
R_{a1}	3
R_{a2}	9
R_{a3}	14



Total Bandwidth Server

- [Server a Priorità Dinamica]

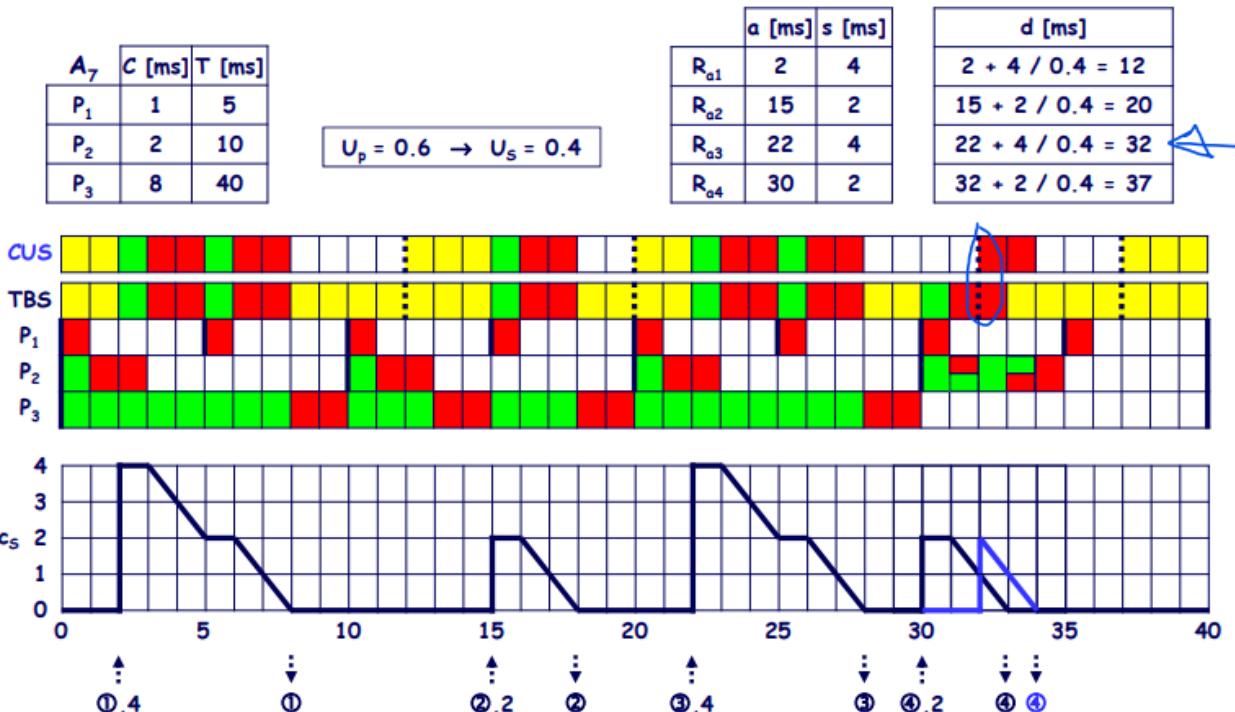
Il server è un processo sporadico con fattore di utilizzazione $U_s \leq 1 - U_p$.

La schedulazione del server segue la stessa strategia utilizzata dagli altri processi (tipicamente EDF, ossia la priorità di ogni processo è dinamica a runtime, ed è inversamente proporzionale alla deadline assoluta, ossia più la deadline è vicina più è alta la priorità).

Quindi ad ogni job release (/arrival) ed all'arrivo di ogni richiesta aperiodica devi verificare qual è il prossimo processo più prioritario.

Solitamente non vengono date le deadline dei processi, in quanto si assumono come quelle di default i loro T!

- La capacità del server inizia a zero e lo stato è WAITING.
- La capacità viene consumata progressivamente solo durante il servizio delle richieste aperiodiche.
- Quando arriva una richiesta (al tempo detto t_{Ra}):
 - la capacità del server c_s si ricarica esattamente di C_{Ra} , ossia il tempo di servizio della richiesta
 - la deadline relativa diventa $d_s = t_{Ra} + c_s / U_s$
- Le richieste che si presentano prima del completamento del servizio vengono accodate. Terminata una richiesta da servire, il server esegue subito le richieste accodate o si mette in WAITING fino all'arrivo della prossima richiesta.
- Se all'istante d_s una o più richieste sono ancora pendenti oppure all'istante di arrivo di una nuova richiesta, vengono calcolati la nuova capacità e deadline, e viene posto P_s in stato READY:
 - Nuovo $c_s = C_{Ra}$
 - Nuova $d_s = \max(d_s, t_{Ra}) + c_s / U_s$
- In caso il valore calcolato sia decimale si considera $d_s' = \lfloor d_s \rfloor$



CUS vs. TBS

PUNTO A)

“Si verifichi se CUS è in grado di fornire le stesse prestazioni di TBS, motivandone le ragioni ed evidenziando in caso contrario nella tabella sottostante i dati relativi alle richieste aperiodiche il cui tempo di completamento del servizio risulta superiore.”

R_{a1}:

R_{a2}: a₂ = 14 > d_{s1} = 11,7

R_{a3}: a₃ = 23 < d_{s2} = 24 TBS non è prioritario in [23-24]

f₁(CUS) = f₁(TBS)

f₂(CUS) = f₂(TBS)

f₃(CUS) = f₃(TBS)

- 1) Completare i campi delle Ra scrivendo il rispettivo tempo di arrivo della richiesta, reperibili dal testo (lasciando la prima Ra₁ in bianco).
 - a. Nella prima riga segnare sempre f₁(CUS) = f₁(TBS)
 - b. Se il tempo di arrivo è < della deadline precedente e TBS è prioritario nell'intervallo [a_i,d_{s_{i-1}}], allora segnare f_i(CUS) > f_i(TBS), altrimenti =
 - c. Se il tempo di arrivo è > della deadline precedente allora segnare f_i(CUS) = f_i(TBS).
 - d. Il significato di tutto questo è cercare di capire se il finishing time di ogni richiesta aperiodica sia uguale o diverso tra CUS e TBS. Per il primo processo ovviamente entrambi i protocolli si comportano allo stesso modo. Successivamente, se l'arrival time di una richiesta avviene prima della deadline della richiesta attualmente servita, allora c'è la possibilità che TBS esegua questa nuova richiesta prima di quando possa eseguire CUS. Se, nell'intervallo tra l'arrivo della nuova richiesta e la deadline, TBS è prioritario rispetto agli altri processi, allora eseguirà sicuramente, e dunque la richiesta verrà servita prima di quanto potrebbe fare CUS (dato che esso esegue solo dopo la deadline).
- 2) Indicare se tale tempo di arrivo è > < = alla deadline della richiesta precedente.
- 3) Indicare a destra il segno > < = :
 - a. Nella prima riga segnare sempre f₁(CUS) = f₁(TBS)
 - b. Se il tempo di arrivo è < della deadline precedente e TBS è prioritario nell'intervallo [a_i,d_{s_{i-1}}], allora segnare f_i(CUS) > f_i(TBS), altrimenti =
 - c. Se il tempo di arrivo è > della deadline precedente allora segnare f_i(CUS) = f_i(TBS).
 - d. Il significato di tutto questo è cercare di capire se il finishing time di ogni richiesta aperiodica sia uguale o diverso tra CUS e TBS. Per il primo processo ovviamente entrambi i protocolli si comportano allo stesso modo. Successivamente, se l'arrival time di una richiesta avviene prima della deadline della richiesta attualmente servita, allora c'è la possibilità che TBS esegua questa nuova richiesta prima di quando possa eseguire CUS. Se, nell'intervallo tra l'arrivo della nuova richiesta e la deadline, TBS è prioritario rispetto agli altri processi, allora eseguirà sicuramente, e dunque la richiesta verrà servita prima di quanto potrebbe fare CUS (dato che esso esegue solo dopo la deadline).
- 4) [se presente nell'esercizio] Riempire la tabella con i valori richiesti. Serve per verificare quanto detto al punto precedente. In particolare:
 - a. TBS:
fi = Quando ho terminato di servire la richiesta asincrona considerata
 - b. CUS:
Se f_i(CUS) = f_i(TBS) inserire sia in fi sia in f_i-ai-Ci un trattino “-”
Se f_i(CUS) > f_i(TBS) occorre calcolare il valore con la formula:
$$f_i(CUS) = f_i(TBS) + ds_{i-1} - a_i - \text{"tempo in READY del TBS in } [a_i; ds_{i-1}]$$

PUNTO B)

“Si verifichi se, indipendentemente dalla effettiva distribuzione temporale e dal tempo di servizio delle richieste aperiodiche, la loro gestione tramite i suddetti server a priorità statica può compromettere o meno la schedulabilità

- a) dei processi P1, P2, P3
- b) di qualunque altro insieme di 3 processi periodici P1', P2', P3' aventi complessivamente lo stesso fattore di utilizzazione del processore ($U_1' + U_2' + U_3' = U_1 + U_2 + U_3$) e periodi $T_1' > T_2' > T_3'$

identificando nell'uno o nell'altro caso il valore massimo che C_s può assumere, a parità di T_s , onde (continuare a) prevenire tale inaccettabile circostanza.”

CASO a)

Completare il diagramma utilizzando il solo protocollo RMPO (o il protocollo indicato esplicitamente nel testo), curandosi solo del T e C dei processi e del server forniti.

Se tutti i processi (e il server?) non hanno missed deadline
⇒ Risposta NO, (!?verificare!?) $C_s \leq C_s$ dato.

Se vi è almeno una missed deadline
⇒ Risposta Sì, $C_s \leq$ valore che sarà sicuramente più piccolo di quello dato. Verificare con il diagramma la sua correttezza.

CASO b)

Indicato con N = numero di processi tranne il server

Nell'esempio:

$$U_s = 2 / 7 = 0.286 \quad U_p = 0.65$$

$$U_{s\ max} = \frac{2}{(1 + \frac{U_p}{N})^N} - 1 = \frac{2}{(1 + \frac{0.65}{4})^4} - 1 = 0.095$$

$$C_{s\ max} = T_s \cdot U_{s\ max} = 7 \cdot 0.095 = 0.665$$

$$U_s = 0.286 > U_{s\ max} = 0.095 \Rightarrow \text{SI}$$

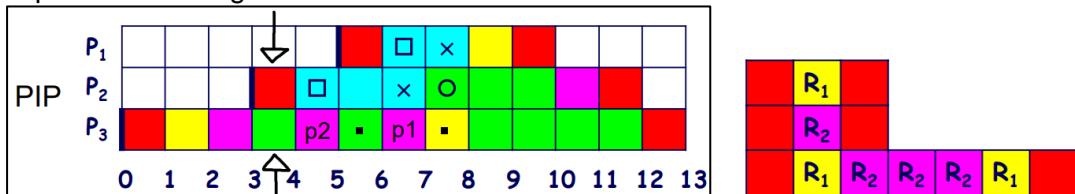
Se $U_s > U_{s\ max}$
⇒ Risposta SI, altrimenti NO

PROBLEMA 3

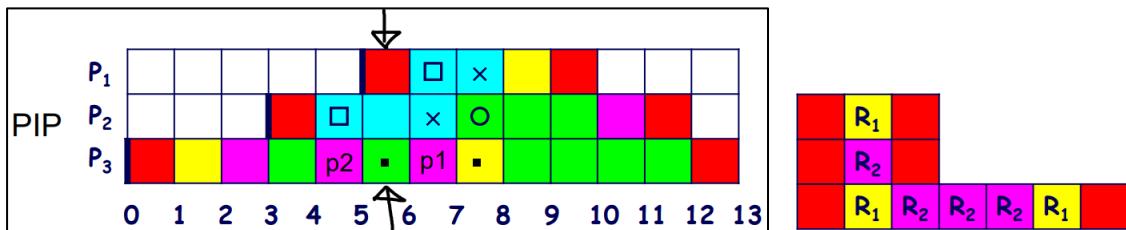
Schedulazione RMPO o DMPO

Ordinare i processi rispettivamente in ordine di periodo (più è piccolo più è prioritario) o in ordine di deadline relativa (più è piccola più è prioritario).

- **Ready.** Il processo è pronto ad eseguire un rosso o risorsa ma un processo di priorità superiore sta eseguendo.



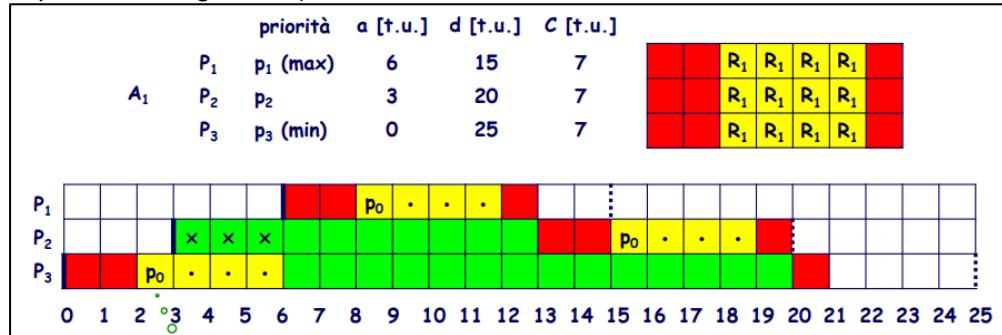
- **Suspended.** Un processo è pronto ad eseguire una risorsa R (NON UN ROSSO) ma un processo di priorità nominalmente inferiore ha attualmente una priorità uguale-superiore e *sta bloccando la risorsa R*.



- **Blocco Diretto.** Un processo cede la sua priorità ad un processo inferiore.
 - Spesso dentro unità Suspended per PIP.
- **Ceiling Block.** Un processo cede la sua priorità ad un processo inferiore perché il primo ha priorità minore al tetto di sistema.
 - Spesso dentro unità Suspended per PCP.
 - Spesso dentro unità Ready per IPCP.
- **Blocco Indiretto (Push-Through).** Un processo non può eseguire perché esegue un processo di priorità nominale inferiore che ora ha priorità superiore.
 - Usato spesso sotto i blocchi diretti o ceiling.
 - Spesso dentro unità Ready.
- **X.** Sostituisce, per sintesi, il simbolo dell'unità adiacente a sinistra.

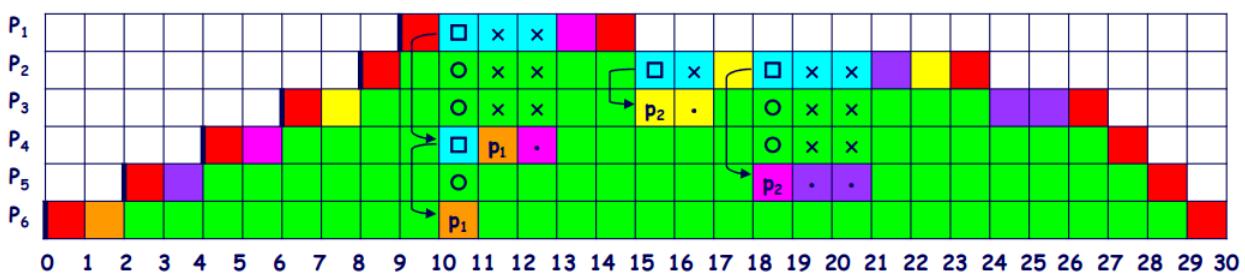
NPCS (NON-PREEMPTIVE CRITICAL SECTION)

- Il primo processo che entra in una sezione critica (solo quando prende una risorsa) non può subire preemption finché non esce dalla sezione critica (acquisisce una priorità "p0" superiore ad ogni altra).



PIP (PRIORITY INHERITANCE)

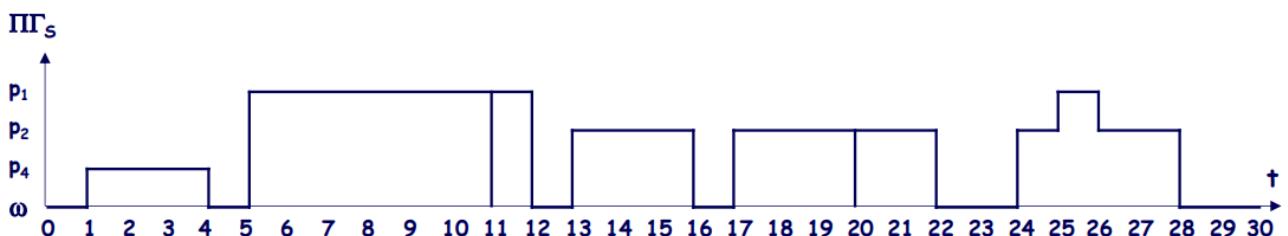
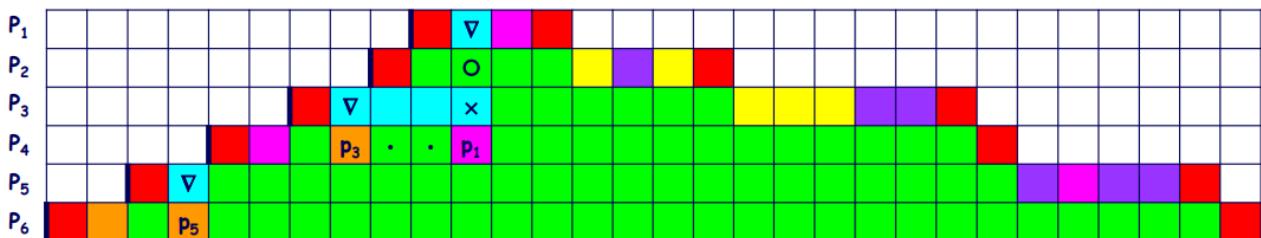
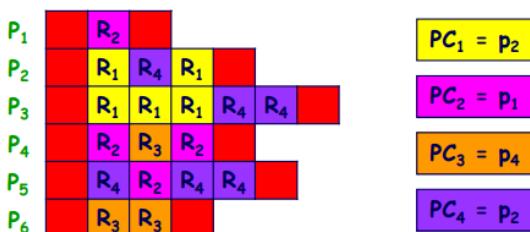
- La priorità corrente di un processo rimane la stessa quando esegue un blocco rosso o quando detiene una risorsa non richiesta da processi di priorità superiore.
- La priorità cambia se un processo vuole accedere ad una risorsa che è già bloccata da un processo di priorità inferiore. Il processo di priorità inferiore **eredita** la priorità del processo superiore che vuole accedere alla risorsa bloccata. *Scrivere la nuova priorità, e scrivere il puntino centrale finché rimane la nuova priorità.*
- Cedere la priorità per eredità è un Blocco Diretto (non vengono mai usati i triangoli).
- Quindi ad un processo è negato l'accesso ad una risorsa solo se essa è già occupata.
- L'eredità è transitiva: se P1 è bloccato da P2 e P2 è bloccato da P3, allora P3 eredita la priorità di P1:



PCP (PRIORITY CEILING)

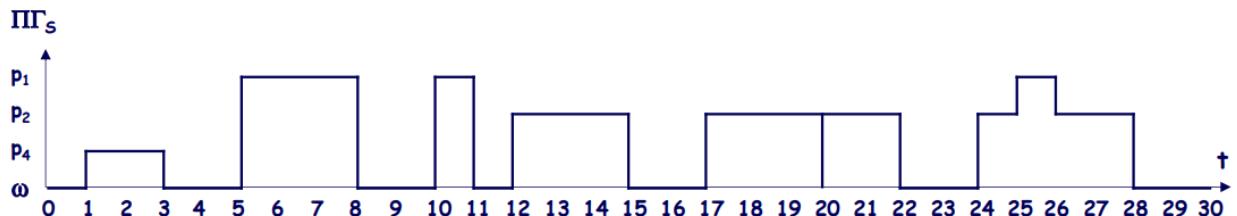
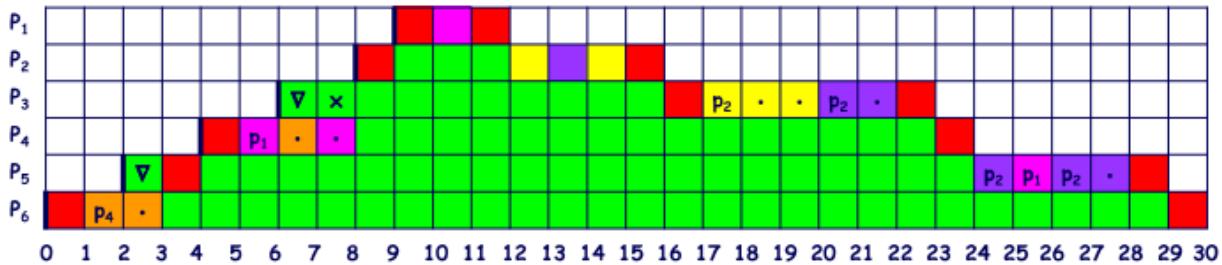
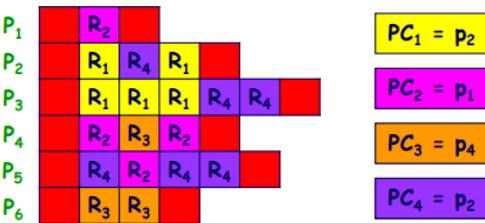
- Identico a PIP: La priorità corrente di un processo rimane la stessa quando esegue un blocco rosso o quando detiene una risorsa non richiesta da processi di priorità superiore.
 - Il tetto di priorità PC_k di una risorsa k è la massima priorità tra i processi che vogliono utilizzare quella risorsa.

Il tetto di priorità di sistema $\Pi\Gamma_s$ è il massimo tra i tetti di priorità delle risorse attualmente bloccate. Quindi il tetto di sistema cambia ogni volta che un processo accede ad una risorsa, e diventa uguale al PC di quella risorsa.
 - Il tetto di sistema inizialmente è a valore w (il minimo) e torna a tale valore quando la risorsa viene rilasciata.
 - L'accesso ad una risorsa è consentito solo se la priorità corrente del processo che vuole accedere è strettamente maggiore del tetto di sistema $\Pi\Gamma_s$.
 - Se un processo non può accedere ad una risorsa perché un processo di priorità inferiore la sta bloccando (e quindi ha alzato il tetto di sistema), il primo processo subisce un Ceiling Block e la sua priorità viene ereditata al secondo processo. Scrivere la nuova priorità, e scrivere il puntino centrale finché rimane la nuova priorità.



IPCP (IMMEDIATE (STACK-BASED) PRIORITY CEILING)

- La priorità corrente di un processo rimane la stessa quando esegue un blocco rosso.
- Identico a PCP: Il tetto di priorità di sistema $\Pi\Gamma_s$ è il massimo tra i tetti di priorità delle risorse attualmente bloccate. *Quindi il tetto di sistema cambia ogni volta che un processo accede ad una risorsa, e diventa uguale al PC di quella risorsa.*
- Quando un processo blocca una risorsa, eredita immediatamente il tetto di priorità della risorsa (che in quel momento è uguale a quello di sistema).
- Una risorsa viene allocata al processo che ne fa richiesta solo se questo ha la priorità maggiore degli altri processi ready, altrimenti riceve un Ceiling Block.
- Processi con la stessa priorità corrente sono schedulati secondo la politica FIFO.



Massimo tempo di blocco

Un Blocco è quando un processo viene rilasciato ma un processo di priorità inferiore viene eseguito perché detiene la risorsa.

Disegnare la tabellina dei consumi delle risorse per ogni processo. Se una risorsa annida altre risorse, per la risorsa che annida bisogna contare anche i blocchi annidati (in più al conteggio del blocco annidato stesso).

- NPCS con RMPO

$$B_i = \max_{j,k} \{Z_{jk} \mid p_j < p_i\} \quad \forall i$$

Per il processo P_i , il blocco è il massimo tempo di esecuzione Z_{jk} della più lunga sezione critica relativa ad una qualsiasi risorsa R_k utilizzata da un qualunque processo di priorità inferiore di quello corrente.

- NPCS con EDF

$$B_i = \max_{j,k} \{Z_{jk} \mid D_j > D_i\} \quad \forall i$$

Per il processo P_i , il blocco è il massimo tempo di esecuzione Z_{jk} della più lunga sezione critica relativa ad una qualsiasi risorsa R_k utilizzata da un qualunque processo con deadline maggiore di quello corrente.

- PIP

$$B_i = \min \left\{ \sum_{j \mid p_j < p_i} \max_k \{Z_{jk} \mid PC_k \geq p_i\}, \sum_{k \mid PC_k \geq p_i} \max_j \{Z_{jk} \mid p_j < p_i\} \right\} \quad \forall i$$

Applico il ragionamento per ogni processo:

Considero le colonne delle risorse usate dal processo e da quelli superiori. Se una risorsa, che compare nel processo preso ora in considerazione, annida un'altra risorsa nei processi successivi, allora quest'altra risorsa annidata va a "sbloccare" la sua stessa colonna del processo preso in considerazione.

Per ogni riga sottostante al processo preso in considerazione, prendo il valore maggiore della riga tra le colonne considerate. Sommo i valori selezionati.

Tra le righe sottostanti al processo preso in considerazione e tra le colonne selezionate, prendo il maggiore di ogni colonna. Sommo i valori selezionati.

Il tempo di blocco per il processo in considerazione è il minimo tra i due valori calcolati precedentemente.

- PCP e IPCP

$$B_i = \max_{j,k} \{Z_{jk} \mid p_j < p_i, PC_k \geq p_i\} \quad \forall i$$

Applico il ragionamento per ogni processo:

Considero le colonne delle risorse usate dal processo e da quelli superiori.

Per ogni riga sottostante al processo preso in considerazione, prendo il valore maggiore tra le colonne considerate.

Analisi di schedulabilità

- **Condizione sufficiente**

- per processi periodici
- ordinati per periodo crescente (priorità decrescente)

1) Calcola per ogni "i" da 1 a N:

$$\sum_{k=1}^i \frac{C_k}{T_k} + \frac{B_i}{T_i}$$

2) È schedulabile con RMPO se ogni valore calcolato è

$$\leq U_{RMPO}(i) = \begin{cases} 1 & \text{a) se i periodi sono in relazione armonica} \\ i(2^{1/i} - 1) & \text{b) in caso contrario} \end{cases}$$

- **Algoritmo di Audsley** (necessario e sufficiente)

- per processi periodici e/o sporadici
- ordinati per deadline relativa crescente (priorità decrescente)

4) La tabella è del tipo:

	P ₁	P ₂	P ₃	P ₄
R _i ⁰				
R _i ¹				
R _i ²				
R _i ³				
R _i ⁴				
R _i ⁵				

In cui ogni riga è una iterazione dei processi. Ad esempio la cella in alto a sinistra è sempre il tempo di risposta R_i⁰

5) Procedi a calcolare colonna per colonna a partire da P₁. Ti fermi a calcolare una colonna appena trovi due tempi di risposta uguali, quindi passi alla colonna successiva.

$$R_i^0 = C_i + B_i \quad \forall i$$

$$R_1^1 = R_1^0$$

$$R_i^n = C_i + B_i + \sum_{k=1}^{i-1} \left\lceil \frac{R_{i-1}^{n-1}}{T_k} \right\rceil C_k \quad i \neq 1$$

6) È schedulabile con DMPO (o con RMPO se D_i = T_i, $\forall i$) se il tempo finale di risposta di ogni processo è **$\leq D_i$** .

Schedulazione EDF

SRP (STACK RESOURCE POLICY)

u_k ($u_k \geq 1$) il numero di unità disponibili per ogni risorsa R_k ($k = 1, \dots, M$),

n_{jk} il numero massimo di unità di R_k richiesto da ciascun processo P_j ($j = 1, \dots, N$),

v_k ($k = 0, \dots, u_k$) il numero di unità di R_k libere nel generico istante t ,

Tabella preemption level dei processi:

Il preemption level di un processo è indirettamente proporzionale alla sua deadline assoluta (ossia la reale): più la deadline è vicina più è alto il pl.

Assegnare i pl ai processi, guardando la deadline relativa fornita.

P_1	$pl_1(\max)$
P_2	pl_2
P_3	pl_3
P_4	pl_4
P_5	$pl_5(\min)$

NOTA BENE:

Il pl di SRP corrisponde sempre alla priorità di EDF (perché entrambi si basano sulla deadline e perché in SRP i blocchi rossi non sono esenti dal confronto col tetto di preemption di sistema). Il pl cambia ad ogni release time. Ma dal momento che tipicamente negli esercizi la deadline relativa di ogni processo è sempre maggiore del rispettivo processo superiore (creando un diagramma che linearmente sale e poi linearmente scende), il pl (e quindi la priorità) rimane sempre costante.

Tabella Tetto di Preemption:

- 1) Ogni riga è una risorsa. Ogni colonna è il numero di unità libere, che assume valori da 0 al valore massimo tra u_1, u_2, \dots, u_k .
- 2) Inserire il trattino “-“ nelle celle in cui $v_k > u_k$, ossia quando le unità libere per una risorsa sono maggiori delle unità disponibili, quindi non ha senso definire tale valore.
- 3) Ragionando a righe, da sinistra a destra, ogni cella è il preemption level più alto tra i processi che hanno un $n_{jk} > v_k$. Ossia guardi la tabella degli “n” fornita, per la risorsa R_k quali sono i processi che la stanno utilizzando e che hanno “ n_{jk} ” strettamente maggiore di “ v_k ”? Ossia quali processi per eseguire hanno bisogno di più di “ v_k ” unità libere? E tra questi, qual è quello con “pl” più alto?
- 4) Se non c’è alcun processo che ha unità richieste (n) maggiore di unità libere (v), ossia se tutti i processi eseguono con quelle unità libere, si scrive ω (che indica la priorità minima del sistema, più bassa anche di quella dell’ultimo processo)

Diagramma temporale:

- Il tetto di preemption di sistema $\Pi\Lambda_s$ inizia a zero. Scrivere separatamente il tetto di ogni risorsa $\Pi\Lambda_k$. Il tetto di sistema è ad ogni istante il maggiore di tutti i tetti. Il tetto della risorsa si abbassa a zero quando la risorsa viene liberata.
- Quando un processo vuole eseguire (a prescindere se eseguire un rosso o se acquisire una risorsa), se il suo pl è strettamente maggiore del tetto di preemption di sistema $\Pi\Lambda_s$ allora può eseguire. Altrimenti si sospende con il simbolo del triangolo.
- Identico a PCP: in risorse annidate, si tiene il preemption level più alto.
- Quando un processo può eseguire una risorsa, il preemption level della risorsa (del suo tetto) diventa il preemption level della tabella di prima, dove la riga è la risorsa utilizzata e la colonna è QUANTE UNITÀ LIBERE DELLA RISORSA SONO RIMASTE DOPO L’ACQUISIZIONE.

Massimo tempo di blocco (SRP)

La tabellina dei massimi tempi di utilizzo di una risorsa è la stessa della prima parte.

$$B_i = \max \{Z_{jk} : p_{lj} < p_{li}, p_{lj} < \Pi \Lambda_k(0) \} \forall i$$

Prendere il massimo tempo di utilizzo di una risorsa (Z_{jk}) purché:

- 1) tale valore sia riferito a un processo con priorità p_{lj} inferiore al processo corrente
- 2) la priorità del processo p_{lj} sia inferiore alla priorità che la risorsa k assume se $v_k = 0$

Ovvero

Tra le righe più basse di quella del processo preso in considerazione, e tra le righe dei processi tali che abbiano p_l inferiore al p_l della risorsa con $v_k = 0$ (prima colonna della tabella compilata all'inizio di SRP), seleziona il valore più grande.

CONSIGLIO: vedi prima qual è il valore più grande di tutti e vedi se rispetta le due condizioni dette sopra; se c'è una parità di valori, inizia a verificare prima quello nella riga più bassa.

Analisi di schedulabilità

- **Condizione sufficiente 1**

- per processi periodici e/o sporadici
- ordinati per deadline relativa crescente:

1) Calcola per ogni "i" da 1 a N:

$$\left(\sum_{k=1}^i \frac{C_k}{D_k} \right) + \frac{B_i}{D_i}$$

2) Se tutti i risultati ottenuti sono ≤ 1 , allora è schedulabile con EDF

- **Condizione sufficiente 2**

- per processi periodici
- ordinati per periodo crescente

1) Calcola per ogni "i" da 1 a N:

$$\left(\sum_{k=1}^i \frac{C_k}{T_k} \right) + \frac{B_i}{T_i}$$

2) Se tutti i risultati ottenuti sono ≤ 1 , allora è schedulabile con EDF

- **Processor Demand** (necessario e sufficiente).

1) Calcolo **Busy Interval**. Proseguì finché $BI^n == BI^{n-1}$. Il Busy Interval finale è l'ultimo calcolato.

$$BI^0 = \sum_{i=1}^N C_i$$

$$BI^n = \sum_{i=1}^N \left\lceil \frac{BI^{n-1}}{T_i} \right\rceil C_i$$

2) Calcoli **Hyperperiod = mcm tra i periodi dei processi**. Verifichi che $BI < \text{Hyperperiod}$.

3) Calcoli

$$t^* = \frac{\sum_{i=1}^N \left(1 - \frac{D_i}{T_i} \right) C_i}{1 - U}$$

4) Calcoli

$$\mathcal{D} = \{d_{ij} \mid d_{ij} = j^* T_i + D_i, d_{ij} < \min(BI, t^*), 1 \leq i \leq N, j \geq 0\}$$

5) Calcoli per ogni elemento "t" nell'insieme precedente e per ogni "i" da 1 a N:

$$\sum_{k=1}^i \left(\left\lfloor \frac{t - D_k}{T_k} \right\rfloor + 1 \right) C_k + \left(\left\lfloor \frac{t - D_i}{T_i} \right\rfloor + 1 \right) B_i$$

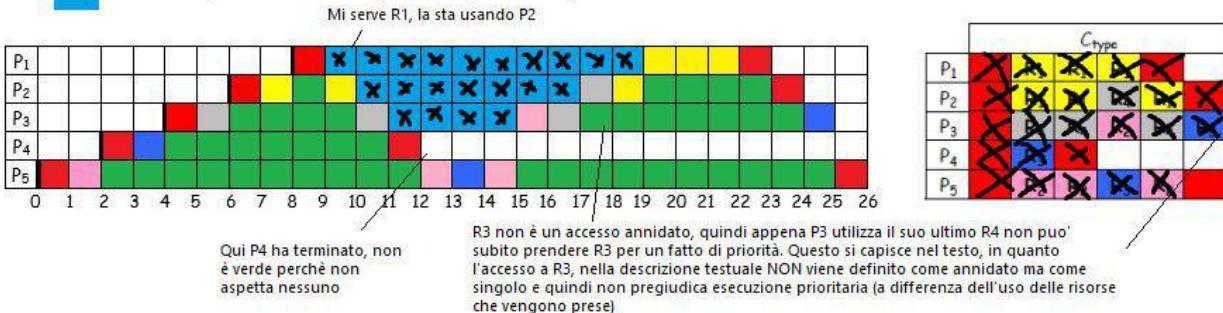
t \ i	1	...	N
t₁			
...			
t_{ultimo}			

6) È schedulabile con EDF se che ogni valore nella prima riga sia minore di t_1, \dots , se ogni valore nell'ultima riga sia minore di t_{ultimo} .

NOP

POSso ESEGUIRE? NO PERCHÉ LA RISORSA LA STA USANDO QUALCUN ALTRO, QUINDI NON FACCIO NIENTE E STO FERMO, ESEGUE QUELLO CON PRIORITA SUBITO INFERIORE.

- Quando un processo è in attesa di un processo di priorità superiore che sta eseguendo
- Quando un processo è in attesa perché un processo di priorità inferiore sta usando una risorsa che ti serve



- Comincia a fare eseguire in ordine di precedenza normale con Priorità $P_1 >$ priorità P_5 ma partendo verso il basso in quanto i processi partono in base alla loro ϕ_i , quindi nell'esempio il primo a partire è P_5 , poi verrà P_4 , poi P_3 etc etc...
- NB: Se un processo inizia con il colore **AZZURRO**, rimane **AZZURRO** finchè non può nuovamente eseguire!!
- Attenzione agli accessi annidati!
Nell'istante [17-18] **P3 non può eseguire usando la risorsa blu R3 in quanto l'accesso annidato è solamente di 4 elementi.**

X X X X

All'istante successivo, P3 non può eseguire **X** in quanto liberando **X** è meno prioritario di P2 che la stava attendendo!